Using MMX™ Instructions in a Fast iDCT Algorithm for MPEG Decoding

Information for Developers and ISVs

From Intel® Developer Services
www.intel.com/IDS

March 1996

CONTENTS

1.0. INTRODUCTION

The Intel Architecture (IA) media extensions include single-instruction, multi-data (SIMD) instructions. This application note presents examples of code that exploit these instructions. Specifically, this document describes an implementation of a two-dimensional (2D) inverse Discrete Cosine Transfer (iDCT) using MMX™ instructions. This transformation is widely used in image compression algorithms; most notably in the Joint Photographic Experts Group (JPEG) and Motion Picture Experts Group (MPEG) standards.

This document focuses on one iDCT algorithm that provides efficient MPEG decoding. The implementation of this algorithm in MMX code, listed in Section 6.0, can be used "as is" according to the guidelines presented in Section 3.1. However, many iDCT algorithms exist. The reader is encouraged to consider the alternative ideas and issues presented in this document, since they have implications for other iDCT algorithms.

MPEG Compression Method

The MPEG compression method has two phases, encoding and decoding. Multimedia images are first encoded, then transmitted, and finally decoded back into a sequence of images by the MPEG player.

The encoding process follows these steps:

1. The input data is transformed using a Discrete Cosine Transform (DCT).
2. The resulting DCT coefficients are quantized.
3. The quantized coefficients are packed efficiently using Huffman tables.

During the decoding process, the MPEG player reverses these steps by unpacking the coefficients, dequantizing them, and applying a 2D iDCT. To achieve a high number of frames per second, this decoding process must be very fast. This document concentrates on the iDCT component of the decoding process.

2.0. iDCT ALGORITHM

The 8x8 two-dimensional iDCT used in MPEG decompression is defined as:

$$s(y,x) = \sum_{u=0}^{7}\sum_{v=0}^{7} \alpha(u)\alpha(v) f(v,u) \cos\frac{(2x+1)\pi u}{16} \cos\frac{(2y+1)\pi v}{16}$$

where the normalization factors, and are defined as:

$$alpha\ () = \frac{1}{2*\sqrt{2}} = \frac{1}{\sqrt{8}}\ for = 0$$

= 1/2 for > 0

Where is either *u* or *v* and $f(v,u)$ is the coefficient matrix.

The solution to this equation contains 64 multiplications and 63 additions for each element of $s(y,x)$, for a total of 4096 multiplications and 4032 additions.

The above equation is equivalent to a summation over *v*, followed by a summation over *u*, as follows:

$$s(y,x) = \sum_{u=0}^{7} \alpha(u) \cos\frac{(2x+1)\pi u}{16} \sum_{v=0}^{7} \alpha(v) f(v,u) \cos\frac{(2y+1)\pi v}{16}$$

This equation is the equivalent to applying a one-dimensional (1D) iDCT eight times on each column of $f(v,u)$ and then applying a 1D iDCT on the rows of the result. Reversing this order by applying a 1D iDCT on the rows first, and then on the columns, gives the same result.

2.1. Selecting a Fast iDCT Algorithm

Many algorithms have been proposed for efficient calculation of the 2D iDCT. Some algorithms are based on efficient 1D iDCTs [4]; some rely on direct analysis of the two-dimensional nature of iDCT [2]; and others combine a 2D prescale with a very efficient 1D iDCT [1]. Some algorithms even take into account the zero coefficients used in the MPEG bit streams to construct statistically efficient iDCT algorithms [3]. Most fast 1D DCT and iDCT algorithms are variants of Lee's Fast DCT algorithm [6], or are based on variants of Winograd's FFT [5].

The following algorithms were evaluated for implementation using MMX instructions:

- Statistic algorithm [3]
- Feig's two-dimensional algorithm [2]
- The LLM algorithm [4]
- The AAN algorithm [1]

In general, statistic algorithms inspect the input data and execute conditional paths in the control flow. The overhead caused by the data inspection and the jump instructions would be too expensive when compared to the speed achievable by other algorithms implemented with MMX instructions.

Although Feig's 2D algorithm [2] reduces the multiplication count, the cost of multiplication in MMX technology is small, so this reduction is not critical. Also, the irregular memory-access pattern of this algorithm is not conducive to efficient implementation using the MMX instruction set.

Both the LLM and AAN algorithms were implemented in MMX assembly code. The LLM algorithm was implemented using accumulation in 32-bit elements, while the AAN algorithm used accumulation in 16-bit elements. The resulting LLM implementation was more accurate, conforming to the iDCT IEEE standard [7]. However, the AAN implementation was much faster. The AAN implementation is presented in this document.

2.2. AAN Algorithm

The AAN algorithm is a one-dimensional, prescaled DCT/iDCT algorithm. First, the eight input coefficients are prescaled by eight prescale values, which requires eight multiplications. Then the algorithm is applied to the prescaled coefficients, which requires five multiplications and 29 additions for the transform. Although the 1D AAN algorithm is less efficient than other 1D algorithms (for example, LLM), when applied to the two-dimensional, 8x8 iDCT, this algorithm takes only 64 multiplications for the prescale, and 80 multiplications and 464 additions for the transform.

3.0. AAN ALGORITHM IMPLEMENTED WITH MMX™ INSTRUCTIONS

The AAN implementation described in this document uses 16-bit data elements, so that four variables can be processed in parallel using packed words. MMX instructions that operate on packed words read or store four words contiguously in memory. So, for an 8x8 matrix, half a row can be read or stored at one time. If the 1D iDCT is applied to the columns of an 8x8 matrix, MMX instructions can operate on four columns at a time. Applying the 1D iDCT to the rows of the matrix is more involved and less efficient.

The AAN algorithm is performed in four steps:

1. Perform an iDCT on the columns of the matrix.
2. Transpose the matrix.
3. Perform a second iDCT.
4. Prescale the input coefficients of the iDCT on the columns of the transposed matrix, which is equivalent to performing an iDCT on the rows of the original matrix.

These steps result in a transposed matrix, which would have to be again transposed to obtain the final result. To prevent this extra step, the input matrix should be transposed initially. The cost of transposing the input is negligible, since the matrix is constructed from the Zig-Zag scan [9]. Therefore, the actual implementation follows these steps:

1. Prescale the input coefficients of the transposed input matrix.
2. Perform an iDCT on the columns of the transposed input matrix, which is equivalent to performing an iDCT on the rows of the final matrix.
3. Transpose the matrix.
4. Perform a second iDCT on the columns of the final matrix.

Another consideration is the limited length of the MMX registers. All the iDCT algorithms mentioned in Section 3.1 are defined mathematically without regard to the size of the accumulator or registers. To ensure adequate precision for operations on 16-bit data elements, the algorithm was analyzed carefully and appropriate precision was assigned during every intermediate stage of the calculation.

Precision was controlled using packed shift instructions, which shift all data elements in a register by the same amount. Shift right instructions were used to prevent overflow of the most significant bit in each intermediate step of the calculation.

3.1. iDCT Routine Interface

The iDCT routine is called from within an assembly module. The routine gets a pointer to an 8x8, 16-bit element matrix; the pointer is located in the ESI register. The matrix should be aligned on an 8-byte boundary. Each data element is actually a 12-bit quantity that is left-adjusted, meaning that the four least significant bits equal zero. The input matrix should be transposed; otherwise, the output matrix must be transposed. The value 0.5 must be added to the DC value, input matrix element [0][0].

The routine returns the same memory array. Therefore, if the original input operands are needed (for example, in test mode), they must be copied before the call to the iDCT routine.

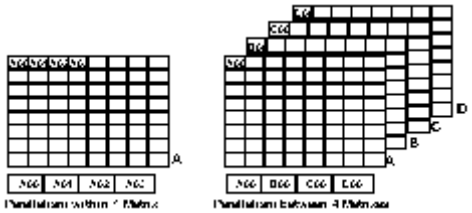
3.2. Optimization Considerations

One standard Pentium® processor optimization technique is code rescheduling to exploit parallelism in an algorithm. Parallelism in the 2D iDCT was approached from two directions, as illustrated in Figure 1:

- Within a single 8x8 iDCT
- Between four 8x8 iDCTs.

In the first approach, data is accessed by rows within the matrix. In the second approach, data from the four matrixes is interleaved to enable efficient use of the MMX instructions.

Figure 1. Single iDCT vs. Four iDCTs



The advantage of the single-iDCT approach is that the interface to an MPEG player is simpler. Its disadvantages are:

- The matrix must be transposed in order to operate on several rows in parallel.
- To prevent overflow, packed shift instructions must be used. Since in a given register, the accuracy of the four data elements varies, the shift count is determined by the worst case among the four elements. This results in an extra loss of accuracy.

The advantages of the four-iDCTs approach are:

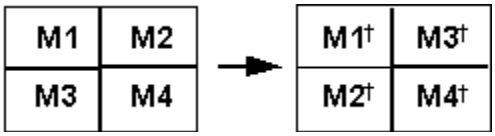
- Matrix transposition is not required.
- To prevent overflow, packed shift instructions must still be used. However, since all the data elements in a register have the same accuracy, there is no extra loss of accuracy to accommodate the worst case among four elements.

The disadvantage of the single-iDCT approach is that, to take advantage of the MMX instructions, the input data from the four matrixes must first be interleaved (see Figure 1). Then, after the transform, the resulting data must be restored to four 8x8 matrixes.

Because of its simplicity, the single-iDCT approach was chosen. Instruction scheduling was done manually to ensure optimal performance.

Register use was carefully considered as well. In most cases, intermediate results were kept in registers; temporary storage to memory was needed in only a few cases. For example, consider the implementation of the matrix transpose. The basic operation is the transpose of 4x4 elements [8], as illustrated in Figure 2.

Figure 2. Matrix Transpose Operation



The 8x8 iDCT requires four of these operations. The sequence of these operations was carefully chosen to save memory stores and loads. First, M4 was transposed, followed by M3. These two results were immediately used to perform the iDCT on the last four columns. Similarly, M2 was transposed, followed by M1. These results were used for the iDCT on the first four columns.

The detailed steps of the matrix transpose algorithm are:

1. Prescale: 16 packed multiplies
2. Column 0: even part
3. Column 0: odd part
4. Column 0: output butterfly
5. Column 1: even part
6. Column 1: odd part
7. Column 1: output butterfly
8. Transpose: M4 part
9. Transpose: M3 part
10. Column 1: even part (after transpose)
11. Column 1: odd part (after transpose)
12. Column 1: output butterfly (after transform)
13. Transpose: M2 parts
14. Transpose: M1 parts
15. Column 0: odd part (after transpose)
16. Column 0: odd part (after transpose)
17. Column 0: output butterfly (after transform)
18. Cleanup

where:

- Column 0 represents the first four columns.

- Even part calculates the part of the iDCT that uses even-indexed elements.
- Odd part calculates the part of the iDCT that uses odd-indexed elements.
- Output butterfly generates the 1D iDCT using the results of the even and odd parts.

During the rescheduling process, instructions from one block were moved to the previous block whenever an empty slot could be filled. This reordering is marked by comments in the code listed in Section 6.0.

4.0. PERFORMANCE GAINS

The cycle count for this implementation of the AAN algorithm, using MMX instructions, is 240 clocks. A direct comparison of this implementation with a scalar implementation of the AAN algorithm would be misleading, since the AAN algorithm is not the fastest scalar implementation of an iDCT. However, the implementation presented here is estimated to be 3 to 3.5 times faster than the general performance of scalar iDCT algorithms.

5.0. REFERENCES

1. Arai, Y., T. Agui, and M. Nakajima, (1988). *A Fast DCT-SQ Scheme for Images*; Trans IEICE, 71, pp. 1095-1097.

2. Feig E., and S. Winograd, (1992). *Fast Algorithms for Discrete Cosine Transform*, IEEE Trans. Signal Proc., 40, pp. 2174-2193.

3. Hung, A.C., and Thy Meng, (1994). *A Fast Statistical Inverse Discrete cosine Transform for Image Compression*, SPIE/IS&Teletronic Imaging ,2187 , pp. 196-205.

4. Loeffler, C., A. Ligtenberg, and C. S. Moschytz, (1989). *Practical Fast 1D DCT Algorithm with Eleven Multiplications*, Proc. ICASSP 1989, pp. 988-991.

5. Winograd S. (1976). *On Computing the Discrete Fourier Transform*, IBM Res. Rep, RC-6291.

6. Lee, B. *A New Algorithm to Compute the Discrete Cosine Transform*, IEEE Trans. Signal Proc., Dec/84, pp. 1243-1245.

7. IEEE standard specification for the implementation of 8x8 iDCT IEEE std 1180-1990

8. MPEG standard, *Coding of Moving Pictures*, ISO/IEC DIS 11172.

6.0. TWO-DIMENSIONAL IDCT CODE LISTING

```
; esi - input and output data pointer
; the input data is tranposed and each 16 bit element in the 8x8 matrix
;is left aligned:
; for example in 11...1110000 format
; If the iDCT is of I macroblock then 0.5 needs to be added to the
;DC Component
; (element[0][0] of the matrix)
.nolist
include iammx.inc                   ; IAMMX Emulator Macros
MMWORD   TEXTEQU <DWORD>
.list
.586
.model flat
_DATA SEGMENT PARA PUBLIC USE32 'DATA'
x0005000200010001         DQ 0005000200010001h
x0040000000000000   DQ 40000000000000h
x5a825a825a825a82         DW 5a82h, 5a82h, 5a82h, 5a82h  ; 23170
x539f539f539f539f         DW 539fh, 539fh, 539fh, 539fh  ; 21407
x4546454645464546         DW 4546h, 4546h, 4546h, 4546h  ; 17734
x61f861f861f861f8         DW 61f8h, 61f8h, 61f8h, 61f8h  ; 25080
scratch1 DQ 0
scratch3 DQ 0
scratch5 DQ 0
scratch7 DQ 0
; for debug only
x0       DQ 0
preSC  DW 16384, 22725, 21407, 19266,  16384, 12873, 8867,  4520
          DW 22725, 31521, 29692, 26722,  22725, 17855, 12299, 6270
          DW 21407, 29692, 27969, 25172,  21407, 16819, 11585, 5906
          DW 19266, 26722, 25172, 22654,  19266, 15137, 10426, 5315
          DW 16384, 22725, 21407, 19266,  16384, 12873, 8867,  4520
          DW 12873, 17855, 16819, 15137,  25746, 20228, 13933, 7103
          DW 17734, 24598, 23170, 20853,  17734, 13933, 9597,  4892
          DW 18081, 25080, 23624, 21261,  18081, 14206, 9785,  4988
_DATA ENDS
_TEXT SEGMENT PARA PUBLIC USE32 'CODE'
COMMENT ^
void idct8x8aan (
    int16 *src_result);
^
public  _idct8x8aan
_idct8x8aan proc near
push     ebp
lea      ecx, [preSC]
mov      ebp, esp
push     esi
mov      esi, DWORD PTR [ebp+8]            ; source
;slot
; column 0: even part
; use V4, V12, V0, V8 to produce V22..V25
movq mm0, mmword ptr [ecx+8*12]            ; maybe the first mul can be done together
                                           ; with the dequantization in iHuff module ?
;slot
pmulhw mm0, mmword ptr [esi+8*12] ; V12
;slot
movq mm1, mmword ptr [ecx+8*4]
;slot
pmulhw mm1, mmword ptr [esi+8*4]  ; V4
;slot
movq mm3, mmword ptr [ecx+8*0]
```

```
psraw mm0, 1                              ; t64=t66
pmulhw mm3, mmword ptr [esi+8*0]  ; V0
;slot
movq mm5, mmword ptr [ecx+8*8]            ; duplicate V4
movq mm2, mm1                           ; added 11/1/96
pmulhw mm5, mmword ptr [esi+8*8]  ; V8
psubsw mm1, mm0                           ; V16
pmulhw mm1, mmword ptr x5a825a825a825a82 ; 23170 ->V18
paddsw mm2, mm0                           ; V17
movq mm0, mm2                             ; duplicate V17
psraw mm2, 1                              ; t75=t82
psraw mm0, 2                              ; t72
movq mm4, mm3                             ; duplicate V0
paddsw mm3, mm5                           ; V19
psubsw mm4, mm5                           ; V20 ;mm5 free
;moved from the block below
movq mm7, mmword ptr [ecx+8*10]
psraw mm3, 1                              ; t74=t81
movq mm6, mm3                             ; duplicate t74=t81
psraw mm4, 2                              ; t77=t79
psubsw mm1, mm0                           ; V21 ; mm0 free
paddsw mm3, mm2                           ; V22
movq mm5, mm1                             ; duplicate V21
paddsw mm1, mm4                           ; V23
movq mmword ptr [esi+8*4], mm3            ; V22
psubsw mm4, mm5                           ; V24; mm5 free
movq mmword ptr [esi+8*12], mm1           ; V23
psubsw mm6, mm2                           ; V25; mm2 free
movq mmword ptr [esi+8*0], mm4            ; V24
;slot
; keep mm6 alive all along the next block
;movq mmword ptr [esi+8*8], mm6           ; V25
; column 0: odd part
; use V2, V6, V10, V14 to produce V31, V39, V40, V41
;moved above
;movq mm7, mmword ptr [ecx+8*10]
pmulhw mm7, mmword ptr [esi+8*10]         ; V10
;slot
movq mm0, mmword ptr [ecx+8*6]
;slot
pmulhw mm0, mmword ptr [esi+8*6]          ; V6
;slot
movq mm5, mmword ptr [ecx+8*2]
movq mm3, mm7                                     ; duplicate V10
pmulhw mm5, mmword ptr [esi+8*2]          ; V2
;slot
movq mm4, mmword ptr [ecx+8*14]
psubsw mm7, mm0                                   ; V26
pmulhw mm4, mmword ptr [esi+8*14]         ; V14
paddsw mm3, mm0                                   ; V29 ; free mm0
movq mm1, mm7                                     ; duplicate V26
psraw mm3, 1                                      ; t91=t94
pmulhw mm7, mmword ptr x539f539f539f539f   ; V33
psraw mm1, 1                                      ; t96
movq mm0, mm5                                     ; duplicate V2
psraw mm4, 2                                      ; t85=t87
paddsw mm5, mm4                                   ; V27
psubsw mm0, mm4                                   ; V28 ; free mm4
movq mm2, mm0                                     ; duplicate V28
psraw mm5, 1                                      ; t90=t93
pmulhw mm0, mmword ptr x4546454645464546  ; V35
psraw mm2, 1                                      ; t97
movq mm4, mm5                                     ; duplicate t90=t93
psubsw mm1, mm2                                   ; V32 ; free mm2
```

```
pmulhw mm1, mmword ptr x61f861f861f861f8   ; V36
psllw mm7, 1                                        ; t107
paddsw mm5, mm3                                     ; V31
psubsw mm4, mm3                                     ; V30 ; free mm3
pmulhw mm4, mmword ptr x5a825a825a825a82   ; V34
nop ;slot
psubsw mm0, mm1                                     ; V38
psubsw mm1, mm7                                     ; V37 ; free mm7
psllw mm1, 1                                        ; t114
;move from the next block
movq mm3, mm6             ; duplicate V25
;move from the next block
movq mm7, mmword ptr [esi+8*4]                      ; V22
psllw mm0, 1                                        ; t110
psubsw mm0, mm5                                     ; V39 (mm5 still needed for next block)
psllw mm4, 2                                        ; t112
;move from the next block
movq mm2, mmword ptr [esi+8*12]   ; V23
psubsw mm4, mm0                                     ; V40
paddsw mm1, mm4                                     ; V41; free mm0
;move from the next block
psllw mm2, 1                                        ; t117=t125
; column 0: output butterfly
;move above
;movq mm3, mm6            ; duplicate V25
;movq mm7, mmword ptr [esi+8*4]                     ; V22
;movq mm2, mmword ptr [esi+8*12]                    ; V23
;psllw mm2, 1                                       ; t117=t125
psubsw mm6, mm1                                     ; tm6
paddsw mm3, mm1                                     ; tm8; free mm1
movq mm1, mm7                                       ; duplicate V22
paddsw mm7, mm5                                     ; tm0
movq mmword ptr [esi+8*8], mm3                      ; tm8; free mm3
psubsw mm1, mm5                                     ; tm14; free mm5
movq mmword ptr [esi+8*6], mm6                      ; tm6; free mm6
movq mm3, mm2                                       ; duplicate t117=t125
movq mm6, mmword ptr [esi+8*0]                      ; V24
paddsw mm2, mm0                                     ; tm2
movq mmword ptr [esi+8*0], mm7                      ; tm0; free mm7
psubsw mm3, mm0                                     ; tm12; free mm0
movq mmword ptr [esi+8*14], mm1                     ; tm14; free mm1
psllw mm6, 1                                        ; t119=t123
movq mmword ptr [esi+8*2], mm2                      ; tm2; free mm2
movq mm0, mm6                                       ; duplicate t119=t123
movq mmword ptr [esi+8*12], mm3                     ; tm12; free mm3
paddsw mm6, mm4                                     ; tm4
;moved from next block
movq mm1, mmword ptr [ecx+8*5]
psubsw mm0, mm4                                     ; tm10; free mm4
;moved from next block
pmulhw mm1, mmword ptr [esi+8*5]          ; V5
;slot
movq mmword ptr [esi+8*4], mm6                      ; tm4; free mm6
;slot
movq mmword ptr [esi+8*10], mm0                     ; tm10; free mm0
;slot
; column 1: even part
; use V5, V13, V1, V9 to produce V56..V59
;moved to prev block
;movq mm1, mmword ptr [ecx+8*5]
;pmulhw mm1, mmword ptr [esi+8*5]         ; V5
movq mm7, mmword ptr [ecx+8*13]
psllw mm1, 1                                        ; t128=t130
pmulhw mm7, mmword ptr [esi+8*13]         ; V13
```

```
movq mm2, mm1                                   ; duplicate t128=t130
movq mm3, mmword ptr [ecx+8*1]
;slot
pmulhw mm3, mmword ptr [esi+8*1]          ; V1
;slot
movq mm5, mmword ptr [ecx+8*9]
psubsw mm1, mm7                                 ; V50
pmulhw mm5, mmword ptr [esi+8*9]          ; V9
paddsw mm2, mm7                                 ; V51
pmulhw mm1, mmword ptr x5a825a825a825a82  ; 23170 ->V52
movq mm6, mm2                                   ; duplicate V51
psraw mm2, 1                                    ; t138=t144
movq mm4, mm3                                   ; duplicate V1
psraw mm6, 2                                    ; t136
paddsw mm3, mm5                                 ; V53
psubsw mm4, mm5                                 ; V54 ;mm5 free
movq mm7, mm3                                   ; duplicate V53
;moved from next block
movq mm0, mmword ptr [ecx+8*11]
psraw mm4, 1                                    ; t140=t142
psubsw mm1, mm6                                 ; V55 ; mm6 free
paddsw mm3, mm2                                 ; V56
movq mm5, mm4                                   ; duplicate t140=t142
paddsw mm4, mm1                                 ; V57
movq mmword ptr [esi+8*5], mm3                  ; V56
psubsw mm5, mm1                                 ; V58; mm1 free
movq mmword ptr [esi+8*13], mm4                 ; V57
psubsw mm7, mm2                                 ; V59; mm2 free
movq mmword ptr [esi+8*9], mm5                  ; V58
;slot
; keep mm7 alive all along the next block
;movq mmword ptr [esi+8*1], mm7                 ; V59
;moved above
;movq mm0, mmword ptr [ecx+8*11]
pmulhw mm0, mmword ptr [esi+8*11]         ; V11
;slot
movq mm6, mmword ptr [ecx+8*7]
;slot
pmulhw mm6, mmword ptr [esi+8*7]          ; V7
;slot
movq mm4, mmword ptr [ecx+8*15]
movq mm3, mm0                                   ; duplicate V11
pmulhw mm4, mmword ptr [esi+8*15]         ; V15
;slot
movq mm5, mmword ptr [ecx+8*3]
psllw mm6,1                                     ; t146=t152
pmulhw mm5, mmword ptr [esi+8*3]          ; V3
paddsw mm0, mm6                                 ; V63
; note that V15 computation has a correction step:
; this is a 'magic' constant that rebiases the results to be closer to the expected result
; this magic constant can be refined to reduce the error even more
; by doing the correction step in a later stage when the number is actually multiplied by 16
paddw mm4, mmword ptr x0005000200010001
psubsw mm3, mm6                                 ; V60 ; free mm6
psraw mm0, 1                                    ; t154=t156
movq mm1, mm3                                   ; duplicate V60
pmulhw mm1, mmword ptr x539f539f539f539f  ; V67
movq mm6, mm5                                   ; duplicate V3
psraw mm4, 2                                    ; t148=t150
;slot
paddsw mm5, mm4                                 ; V61
psubsw mm6, mm4                                 ; V62 ; free mm4
movq mm4, mm5                                   ; duplicate V61
psllw mm1, 1                                    ; t169
```

```
paddsw mm5, mm0                                  ; V65 -> result
psubsw mm4, mm0                                  ; V64 ; free mm0
pmulhw mm4, mmword ptr x5a825a825a825a82   ; V68
psraw mm3, 1                                     ; t158
psubsw mm3, mm6                                  ; V66
movq mm2, mm5                                    ; duplicate V65
pmulhw mm3, mmword ptr x61f861f861f861f8   ; V70
psllw mm6, 1                                     ; t165
pmulhw mm6, mmword ptr x4546454645464546   ; V69
psraw mm2, 1                                     ; t172
;moved from next block
movq mm0, mmword ptr [esi+8*5]                   ; V56
psllw mm4, 1                                     ; t174
;moved from next block
psraw mm0, 1                                     ; t177=t188
nop ; slot
psubsw mm6, mm3                                  ; V72
psubsw mm3, mm1                                  ; V71 ; free mm1
psubsw mm6, mm2                                  ; V73 ; free mm2
;moved from next block
psraw mm5, 1                                     ; t178=t189
psubsw mm4, mm6                                  ; V74
;moved from next block
movq mm1, mm0                                    ; duplicate t177=t188
paddsw mm3, mm4                                  ; V75
;moved from next block
paddsw mm0, mm5                                  ; tm1
;location
;  5 - V56
; 13 - V57
;  9 - V58
;  X - V59, mm7
;  X - V65, mm5
;  X - V73, mm6
;  X - V74, mm4
;  X - V75, mm3
; free mm0, mm1 & mm2
;move above
;movq mm0, mmword ptr [esi+8*5]                  ; V56
;psllw mm0, 1                                    ; t177=t188 ! new !!
;psllw mm5, 1                                    ; t178=t189 ! new !!
;movq mm1, mm0                                   ; duplicate t177=t188
;paddsw mm0, mm5                                 ; tm1
movq mm2, mmword ptr [esi+8*13]                  ; V57
psubsw mm1, mm5                                  ; tm15; free mm5
movq mmword ptr [esi+8*1], mm0                   ; tm1; free mm0
psraw mm7, 1                                     ; t182=t184 ! new !!
;save the store as used directly in the transpose
;movq mmword ptr [esi+8*15], mm1          ; tm15; free mm1
movq mm5, mm7                            ; duplicate t182=t184
psubsw mm7, mm3                                  ; tm7
paddsw mm5, mm3                                  ; tm9; free mm3
;slot
movq mm0, mmword ptr [esi+8*9]                   ; V58
movq mm3, mm2                                    ; duplicate V57
movq mmword ptr [esi+8*7], mm7                   ; tm7; free mm7
psubsw mm3, mm6                                  ; tm13
paddsw mm2, mm6                                  ; tm3 ; free mm6
; moved up from the transpose
movq mm7, mm3
; moved up from the transpose
punpcklwd mm3, mm1
movq mm6, mm0                                    ; duplicate V58
movq mmword ptr [esi+8*3], mm2                   ; tm3; free mm2
```

```
paddsw mm0, mm4                                 ; tm5
psubsw mm6, mm4                                 ; tm11; free mm4
; moved up from the transpose
punpckhwd mm7, mm1
movq mmword ptr [esi+8*5], mm0                  ; tm5; free mm0
; moved up from the transpose
movq mm2, mm5
; transpose - M4 part
;  ---------       ---------
; | M1 | M2 |     | M1'| M3'|
;  ---------  -->  ---------
; | M3 | M4 |     | M2'| M4'|
;  ---------       ---------
; Two alternatives: use full mmword approach so the following code can be
; scheduled before the transpose is done without stores, or use the faster
; half mmword stores (when possible)
movdf dword ptr [esi+8*9+4], mm3  ; MS part of tmt9
punpcklwd mm5, mm6
movdf dword ptr [esi+8*13+4], mm7 ; MS part of tmt13
punpckhwd mm2, mm6
movdf dword ptr [esi+8*9], mm5            ; LS part of tmt9
punpckhdq mm5, mm3                              ; free mm3
movdf dword ptr [esi+8*13], mm2           ; LS part of tmt13
punpckhdq mm2, mm7                              ; free mm7
; moved up from the M3 transpose
movq mm0, mmword ptr [esi+8*8]
;slot
; moved up from the M3 transpose
movq mm1, mmword ptr [esi+8*10]
; moved up from the M3 transpose
movq mm3, mm0
; shuffle the rest of the data, and write it with 2 mmword writes
movq mmword ptr [esi+8*11], mm5   ; tmt11
; moved up from the M3 transpose
punpcklwd mm0, mm1
movq mmword ptr [esi+8*15], mm2           ; tmt15
; moved up from the M3 transpose
punpckhwd mm3, mm1
; transpose - M3 part
; moved up to previous code section
;movq mm0, mmword ptr [esi+8*8]
;movq mm1, mmword ptr [esi+8*10]
;movq mm3, mm0
;punpcklwd mm0, mm1
;punpckhwd mm3, mm1
movq mm6, mmword ptr [esi+8*12]
;slot
movq mm4, mmword ptr [esi+8*14]
movq mm2, mm6
; shuffle the data and write out the lower parts of the transposed in 4 dwords
punpcklwd mm6, mm4
movq mm1, mm0
punpckhdq mm1, mm6
movq mm7, mm3
punpckhwd mm2, mm4                              ; free mm4
;slot
punpckldq mm0, mm6                              ; free mm6
;slot
;moved from next block
movq mm4, mmword ptr [esi+8*13]                 ; tmt13
punpckldq mm3, mm2
punpckhdq mm7, mm2                              ; free mm2
;moved from next block
movq mm5, mm3                                   ; duplicate tmt5
```

```
; column 1: even part (after transpose)
;moved above
;movq mm5, mm3                                  ; duplicate tmt5
;movq mm4, mmword ptr [esi+8*13]          ; tmt13
psubsw mm3, mm4                                 ; V134
;slot
pmulhw mm3, mmword ptr x5a825a825a825a82  ; 23170 ->V136
;slot
movq mm6, mmword ptr [esi+8*9]                  ; tmt9
paddsw mm5, mm4                                 ; V135 ; mm4 free
movq mm4, mm0                                   ; duplicate tmt1
paddsw mm0, mm6                                 ; V137
psubsw mm4, mm6                                 ; V138 ; mm6 free
psllw mm3, 2                                    ; t290
psubsw mm3, mm5                                 ; V139
movq mm6, mm0                                   ; duplicate V137
paddsw mm0, mm5                                 ; V140
movq mm2, mm4                                   ; duplicate V138
paddsw mm2, mm3                                 ; V141
psubsw mm4, mm3                                 ; V142 ; mm3 free
movq mmword ptr [esi+8*9], mm0                  ; V140
psubsw mm6, mm5                                 ; V143 ; mm5 free
;moved from next block
movq mm0, mmword ptr[esi+8*11]                  ; tmt11
;slot
movq mmword ptr [esi+8*13], mm2           ; V141
;moved from next block
movq mm2, mm0                                   ; duplicate tmt11
; column 1: odd part (after transpose)
;moved up to the prev block
;movq mm0, mmword ptr[esi+8*11]                 ; tmt11
;movq mm2, mm0                                  ; duplicate tmt11
movq mm5, mmword ptr[esi+8*15]                  ; tmt15
psubsw mm0, mm7                                 ; V144

movq mm3, mm0                                   ; duplicate V144
paddsw mm2, mm7                                 ; V147 ; free mm7
pmulhw mm0, mmword ptr x539f539f539f539f  ; 21407-> V151
movq mm7, mm1                                   ; duplicate tmt3
paddsw mm7, mm5                                 ; V145
psubsw mm1, mm5                                 ; V146 ; free mm5
psubsw mm3, mm1                                 ; V150
movq mm5, mm7                                   ; duplicate V145
pmulhw mm1, mmword ptr x4546454645464546  ; 17734-> V153
psubsw mm5, mm2                                 ; V148
pmulhw mm3, mmword ptr x61f861f861f861f8  ; 25080-> V154
psllw mm0, 2                                    ; t311
pmulhw mm5, mmword ptr x5a825a825a825a82  ; 23170-> V152
paddsw mm7, mm2                                 ; V149 ; free mm2
psllw mm1, 1                                    ; t313
nop ; slot
;without the nop above - freeze here for one clock
;the nop cleans the mess a little bit
movq mm2, mm3                                   ; duplicate V154
psubsw mm3, mm0                                 ; V155 ; free mm0
psubsw mm1, mm2                                 ; V156 ; free mm2
;moved from the next block
movq mm2, mm6                                   ; duplicate V143
;moved from the next block
movq mm0, mmword ptr[esi+8*13]    ; V141
psllw mm1, 1                                    ; t315
psubsw mm1, mm7                                 ; V157 (keep V149)
psllw mm5, 2                                    ; t317
psubsw mm5, mm1                                 ; V158
```

```
psllw mm3, 1                                    ; t319
paddsw mm3, mm5                                 ; V159
;slot
; column 1: output butterfly (after transform)
;moved to the prev block
;movq mm2, mm6                                  ; duplicate V143
;movq mm0, mmword ptr[esi+8*13]   ; V141
psubsw mm2, mm3                                 ; V163
paddsw mm6, mm3                                 ; V164 ; free mm3
movq mm3, mm4                                   ; duplicate V142
psubsw mm4, mm5                                 ; V165 ; free mm5
movq mmword ptr scratch7, mm2                   ; out7
psraw mm6, 4
psraw mm4, 4
paddsw mm3, mm5                                 ; V162
movq mm2, mmword ptr[esi+8*9]                   ; V140
movq mm5, mm0                                   ; duplicate V141
;in order not to perculate this line up, we read [esi+8*9] very near to this location
movq mmword ptr [esi+8*9], mm6                  ; out9
paddsw mm0, mm1                                 ; V161
movq mmword ptr scratch5, mm3                   ; out5
psubsw mm5, mm1                                 ; V166 ; free mm1
movq mmword ptr[esi+8*11], mm4                  ; out11
psraw mm5, 4
movq mmword ptr scratch3, mm0                   ; out3
movq mm4, mm2                                   ; duplicate V140
movq mmword ptr[esi+8*13], mm5                  ; out13
paddsw mm2, mm7                                 ; V160
;moved from the next block
movq mm0, mmword ptr [esi+8*1]
psubsw mm4, mm7                                 ; V167 ; free mm7
;moved from the next block
movq mm7, mmword ptr [esi+8*3]
psraw mm4, 4
movq mmword ptr scratch1, mm2                   ; out1
;moved from the next block
movq mm1, mm0
movq mmword ptr[esi+8*15], mm4                  ; out15
;moved from the next block
punpcklwd mm0, mm7
; transpose - M2 parts
;moved up to the prev block
;movq mm0, mmword ptr [esi+8*1]
;movq mm7, mmword ptr [esi+8*3]
;movq mm1, mm0
;punpcklwd mm0, mm7
movq mm5, mmword ptr [esi+8*5]
punpckhwd mm1, mm7
movq mm4, mmword ptr [esi+8*7]
movq mm3, mm5
; shuffle the data and write out the lower parts of the trasposed in 4 dwords
movdf dword ptr [esi+8*8], mm0            ; LS part of tmt8
punpcklwd mm5, mm4
movdf dword ptr [esi+8*12], mm1           ; LS part of tmt12
punpckhwd mm3, mm4
movdf dword ptr [esi+8*8+4], mm5          ; MS part of tmt8
punpckhdq mm0, mm5                                ; tmt10
movdf dword ptr [esi+8*12+4], mm3                 ; MS part of tmt12
punpckhdq mm1, mm3                                ; tmt14
; transpose - M1 parts
movq mm7, mmword ptr [esi]
;slot
movq mm2, mmword ptr [esi+8*2]
movq mm6, mm7
```

```
movq mm5, mmword ptr [esi+8*4]
punpcklwd mm7, mm2
movq mm4, mmword ptr [esi+8*6]
punpckhwd mm6, mm2 ; free mm2
movq mm3, mm5
punpcklwd mm5, mm4
punpckhwd mm3, mm4                                  ; free mm4
movq mm2, mm7
movq mm4, mm6
punpckldq mm7, mm5                                  ; tmt0
punpckhdq mm2, mm5                                  ; tmt2 ; free mm5
;slot
; shuffle the rest of the data, and write it with 2 mmword writes
punpckldq mm6, mm3                                  ; tmt4
;move from next block
movq mm5, mm2                                       ; duplicate tmt2
punpckhdq mm4, mm3                                  ; tmt6 ; free mm3
;move from next block
movq mm3, mm0                                       ; duplicate tmt10
; column 0: odd part (after transpose)
;moved up to prev block
;movq mm3, mm0                                      ; duplicate tmt10
;movq mm5, mm2                                      ; duplicate tmt2
psubsw mm0, mm4                                     ; V110
paddsw mm3, mm4                                     ; V113 ; free mm4
movq mm4, mm0                                       ; duplicate V110
paddsw mm2, mm1                                     ; V111
pmulhw mm0, mmword ptr x539f539f539f539f   ; 21407-> V117
psubsw mm5, mm1                                     ; V112 ; free mm1
psubsw mm4, mm5                                     ; V116
movq mm1, mm2                                       ; duplicate V111
pmulhw mm5, mmword ptr x4546454645464546   ; 17734-> V119
psubsw mm2, mm3                                     ; V114
pmulhw mm4, mmword ptr x61f861f861f861f8   ; 25080-> V120
paddsw mm1, mm3                                     ; V115 ; free mm3
pmulhw mm2, mmword ptr x5a825a825a825a82   ; 23170-> V118
psllw mm0, 2                                        ; t266
movq mmword ptr[esi+8*0], mm1                       ; save V115
psllw mm5, 1                                        ; t268
psubsw mm5, mm4                                     ; V122
psubsw mm4, mm0                                     ; V121 ; free mm0
psllw mm5, 1                                        ; t270
;slot
psubsw mm5, mm1                                     ; V123 ; free mm1
psllw mm2, 2                                        ; t272
psubsw mm2, mm5                                     ; V124 (keep V123)
psllw mm4, 1                                        ; t274
movq mmword ptr[esi+8*2], mm5                       ; save V123 ; free mm5
paddsw mm4, mm2                                     ; V125 (keep V124)
; column 0: even part (after transpose)
movq mm0, mmword ptr[esi+8*12]                      ; tmt12
movq mm3, mm6                                       ; duplicate tmt4
psubsw mm6, mm0                                     ; V100
paddsw mm3, mm0                                     ; V101 ; free mm0
pmulhw mm6, mmword ptr    x5a825a825a825a82         ; 23170 ->V102
movq mm5, mm7                                       ; duplicate tmt0
movq mm1, mmword ptr[esi+8*8]                       ; tmt8
;slot
paddsw mm7, mm1                                     ; V103
psubsw mm5, mm1                                     ; V104 ; free mm1
movq mm0, mm7                                       ; duplicate V103
psllw mm6, 2                                        ; t245
paddsw mm7, mm3                                     ; V106
movq mm1, mm5                                       ; duplicate V104
```

```
psubsw mm6, mm3                                 ; V105
psubsw mm0, mm3                                 ; V109; free mm3
paddsw mm5, mm6                                 ; V107
psubsw mm1, mm6                                 ; V108 ; free mm6
; column 0: output butterfly (after transform)
movq mm3, mm1                                   ; duplicate V108
paddsw mm1, mm2                                 ; out4
psraw mm1, 4
psubsw mm3, mm2                                 ; out10 ; free mm2
psraw mm3, 4
movq mm6, mm0                                   ; duplicate V109
movq mmword ptr[esi+8*4], mm1                   ; out4 ; free mm1
psubsw mm0, mm4                                 ; out6
movq mmword ptr[esi+8*10], mm3                  ; out10 ; free mm3
psraw mm0, 4
paddsw mm6, mm4                                 ; out8 ; free mm4
movq mm1, mm7                                   ; duplicate V106
movq mmword ptr[esi+8*6], mm0                   ; out6 ; free mm0
psraw mm6, 4
movq mm4, mmword ptr[esi+8*0]                   ; V115
;slot
movq mmword ptr[esi+8*8], mm6                   ; out8 ; free mm6
movq mm2, mm5    ; duplicate V107
movq mm3, mmword ptr[esi+8*2]                   ; V123
paddsw mm7, mm4                                 ; out0
;moved up from next block
movq mm0, mmword ptr scratch3
psraw mm7, 4
;moved up from next block
movq mm6, mmword ptr scratch5
psubsw mm1, mm4                                 ; out14 ; free mm4
paddsw mm5, mm3                                 ; out2
psraw mm1, 4
movq mmword ptr[esi], mm7                ; out0 ; free mm7
psraw mm5, 4
movq mmword ptr[esi+8*14], mm1                  ; out14 ; free mm1
psubsw mm2, mm3                                 ; out12 ; free mm3
movq mmword ptr[esi+8*2], mm5                   ; out2 ; free mm5
psraw mm2, 4
;moved up to the prev block
movq mm4, mmword ptr scratch7
;moved up to the prev block
psraw mm0, 4
movq mmword ptr[esi+8*12], mm2                  ; out12 ; free mm2
;moved up to the prev block
psraw mm6, 4
;move back the data to its correct place
;moved up to the prev block
;movq mm0, mmword ptr scratch3
;movq mm6, mmword ptr scratch5
;movq mm4, mmword ptr scratch7
;psraw mm0, 4
;psraw mm6, 4
movq mm1, mmword ptr scratch1
psraw mm4, 4
movq mmword ptr [esi+8*3], mm0           ; out3
psraw mm1, 4
movq mmword ptr [esi+8*5], mm6           ; out5
;slot
movq mmword ptr [esi+8*7], mm4           ; out7
;slot
movq mmword ptr [esi+8*1], mm1           ; out1
;slot
emms
```

```
pop esi
pop ebp
ret     0
_idct8x8aan ENDP
_TEXT ENDS
END
```